Міністерство освіти і науки України

ОДЕСЬКА НАЦІОНАЛЬНА АКАДЕМІЯ ЗВ'ЯЗКУ ім. О.С. ПОПОВА

Кафедра комп'ютерно-інтегрованих технологічних процесів і виробництв

Методичні вказівки до виконання лабораторної роботи "Програмування ПЛК. Вивчення мови структурований текст (ST)"

з дисципліни
"Мікропроцесорні та програмні засоби автоматизації"

Модуль 1 Керування процесом за допомогою цифрових пристроїв у засобах автоматизації.
Побудова систем управління на контролерах та мікроконтролерах різних типів

для бакалаврів галузі знань 15
"Автоматизація та приладобудування"
за спеціальністю 151
"Автоматизація та комп'ютерно-інтегровані технології"

Одеса 2017

УДК 681.5.004.9 План НМВ 2017 р.

Рецензент – д.т.н., проф. Лісовий І.П.

Тігарєв А.М. Методичні вказівки до виконання лабораторної роботи “Програмування ПЛК. Вивчення мови структурований текст (ST)”/Тігарєв А.М. – Одеса: ОНАЗ ім. О.С. Попова, 2017. – 15 с.

У методичних вказівках розглянуто реалізацію автоматизації ділянок технологічного процесу за допомогою мови структурований текст (ST). Названо вимоги до виконання програм мовою програмування ST за стандартом МЕК 61131-3.

Ухвалено
на кафедрі комп’ютерно-інтегрованих
технологічних процесів і виробництв
і рекомендовано до друку
Протокол № 4
від 10.11.2017 р.

Затверджено
методичною радою
академії зв’язку
Протокол № 5
від 28.02. 2017 р.

ЗМІСТ

Лабораторна робота № 5

ПРОГРАМУВАННЯ ПЛК. ВИВЧЕННЯ МОВИ СТРУКТУРОВАНИЙ ТЕКСТ (ST)

1 Мета роботи

Метою роботи є ознайомлення з принципами програмування мовою структурований текст **(ST)** за стандартом **МЕК 61131-3** для програмованих логічних контролерів при розробці окремих вузлів систем керування, а також набуття навичок роботи в інтегрованому середовищі розробки програм **CoDeSys**.

2 Основні положення

Мова **ST** є **Паскалє**-подібною мовою, що дозволяє реалізувати складні алгоритми з умовами переходу, різноманітних циклів, спеціальних математичних функцій та інше. Мова **ST** є подальшим розвитком мови **PL7-3/Litteral**, до якої додані нові функції. Мова **ST** найбільш звична для професійних програмістів. Вона значно використовується при програмуванні ПЛК, які не мають бібліотеки програм, за допомогою набору необхідної програми на персональному комп'ютері, підключеному до ПЛК. Основою **ST**-програми служать вирази. **ST** являє собою набір інструкцій високого рівня, які можуть використовуватися в умовних операторах **("IF…THEN…ELSE")** і в циклах **(WHILE…DO)...** Розглянемо основні конструкції мови **ST**.

2.1 Вирази

Вираз – це конструкція, яка повертає певне значення після його обчислення.

Вираз складається з операторів і операндів. Операндом може бути константа, змінна, функціональний блок або інший вираз. Кожний вираз обов'язково закінчується крапкою з комою «**;**». Стандартні оператори у виразах **ST** мають символьне подання, наприклад, математичні дії: **+, -, *, /,** операції порівняння й т. ін. Наприклад:

iVarl: = 1 + iVar2 / ABS(iVar2);

Обчислення виразів виконується відповідно до правил пріоритету. Оператор з найвищим пріоритетом виконується першим, оператор з більш низьким пріоритетом – другим і т. ін., поки не будуть виконані всі оператори. Оператори з однаковим пріоритетом виконуються зліва направо.

У наступній таблиці наданий список **ST** операторів, розташованих у порядку пріоритету.

№	Операція	Позначення	Пріоритет
1	Вираження в дужках	(Вираження)	Найвищий
2	Виклик функції	**Ім'я функції (список параметрів)**	1
3	Зведення в ступінь	**EXPT**	2
4	Заміна знаку	**–**	3
5	Числове доповнення	**NOT**	4
6	Розподіл	**/**	5
7	Залишок від розподілу	**MOD**	6
8	Додавання	**+**	7
9	Віднімання	**–**	8
10	Порівняння	**< , > ,<=, >=**	9
11	Нерівність	**< >**	10
12	Рівність	**=**	11
13	Логічне **І**	**AND**	12
14	Логічне, що виключає **АБО**	**XOR**	13
15	Логічне **АБО**	**OR**	Найнижчий

Крім операторів елементи виразів можна відокремлювати пробілами й табуляціями для кращого сприйняття. У текст можуть бути уведені коментарі, які записуються між символами **(* *)**. Скрізь, де припустимі пасивні роздільники, можна вставляти й коментарі:

iVarl := 1 + (*одержати знак***) iVar2/ABS(iVar2); (***перевірка на **0** була вище***)**

Кілька виразів можна записати підряд в один рядок. Довгі вирази можна перенести на наступний рядок. Перенесення рядка рівноцінне пасивному роздільнику.

Тип виразів визначається типом результату обчислень:

bAlarm := byInpl>byInp2 AND byInpl + byInp2 <> 0 OR bAlarm 2;

Вираз може включати інші вирази, які укладені в дужки. Вирази, які укладені в дужки, обчислюється в першу чергу.

При складанні виразів обов'язково необхідно враховувати можливий діапазон зміни значень і типи змінних. Помилки, пов'язані з переповненням, виникають у процесі виконання й не можуть бути виявлені транслятором.

В **ST** припустимо використати порожній вираз, що складається з крапки з комою «**;**». Для крапки з комою транслятор не генерує ніякого коду. Якщо випадково поставити зайву « **;** », це не викликає помилки. Єдине осмислене застосування порожнього виразу – це забезпечення правильності конструкцій мови **ST**. Наприклад, може знадобитися відтранслювати проект, що містить ще

не реалізований **POU**. Для коректної трансляції досить написати в тілі **POU** один порожній оператор. Ще один приклад, де порожній оператор виявляється досить до речі, це умова **IF**, яка не підтримує розділ **THEN**.

2.2 Оператор присвоювання

Перед оператором присвоювання «**:=**» перебуває операнд (змінна або адреса), якому привласнюється значення виразу, який розташований після оператора присвоювання.

Приклад:

Var1:= Var2 * 10;

Після виконання цієї операції **Var1** набуває значення в десять разів більше, ніж **Var2**.

2.3 Виклик функціонального блока в ST

Функціональний блок викликається за допомогою імені екземпляра функціонального блока й списку вхідних параметрів з присвоюванням даних у круглих дужках. У наступному прикладі викликається таймер з параметрами **IN** й **PT**. Значення вихідної змінної **Q** привласнюється змінної **A**.

До вихідної змінної, як й в **IL**, можна звернутися за допомогою імені екземпляра функціонального блока, крапки та наступним за ними іменем вихідної змінної:

CMD_TMR (IN: = %IX5, PT: = 300);
A: = CMD_TMR.Q

2.4 Інструкція IF

Інструкція IF являє собою оператор вибору, що дозволяє виконати різні групи виразів залежно від умов, які виражені логічними виразами. Повний синтаксис оператора **IF** (якщо) виглядає так:

IF <логічний вираз **IF**>
THEN
<вираз **IF**>
[
ELSIF <логічний вираз **ELSEIF1**>
THEN
< вираз **ELSEIF 1**> **;**

…
ELSIF <логічний вираз **ELSEIF n**>
THEN
< вираз **ELSEIF n**> **;**
ELSE
< вираз **ELSE**> **;**

]
END_IF

Якщо **<**логічний вираз **IF> TRUE (ІСТИНА)**, то виконуються вирази першої групи – <вирази **IF**>. Інші вирази пропускаються, альтернативні умови не перевіряються. Застосовувати оператор присвоювання **«:=»** в <логічному виразі **IF**> не можна.

Частина конструкції у квадратних дужках є необов'язковою й може бути відсутня.

Якщо **<**логічний вираз **IF> FALSE (НЕПРАВДА)**, то одна за одною перевіряються умови **ELSIF**. Перша істинна умова приведе до виконання відповідної групи виразів. Інші умови **ELSIF** аналізуватися не будуть. Груп **ELSIF** може бути декілька або не бути зовсім.

Якщо всі логічні вирази дали помилковий результат, то виконуються вирази групи **ELSE**, якщо вона є. Якщо групи **ELSE** нема, то не виконується нічого.

У найпростішому випадку оператор **IF** містить тільки одну умову:

Приклад:

IF temp < 17
THEN heating_on := TRUE;
ELSE heating_on: = FALSE;
END_IF

У цьому прикладі нагрівання (**heating**) вмикається, коли температура опуститься нижче **17**^{o} градусів, інакше воно залишиться виключеним.

2.5 Інструкція CASE

Інструкція **CASE** є оператор множинного вибору **CASE**, який дозволяє виконати різні групи виразів залежно від значення однієї цілочисельної змінної або виразу.

Синтаксис:

CASE < цілочисельний вираз> **OF**
<значення **1**>**:**
<вираз **1**>**;**
<значення **2**> **,** <значення **3**> **:**
<вираз **3**> **;**
<значення **4**>**..**<значення **5**> **:**
<вираз **4**> **;**
…
[
ELSE
<вираз **ELSE**>
]
END CASE

Інструкція **CASE** виконується відповідно до наступних правил.

Якщо значення виразу збігається із заданою константою, то виконується відповідна група виразів. Інші умови не аналізуються **(**<значення **1**>**:** <вирази **1**> **;)**.

Якщо кілька значень констант повинні відповідати одній групі виразів, їх можна перелічити через кому **(**<значення **2**> **,** <значення **3**> **:** <вирази **3**> **;)**.

Діапазон значень можна визначити через двокрапку **(**<значення **4**>**..**<значення **5**> **:** <вирази **4**> **;)**.

Група виразів **ELSE** є необов'язковою. Вона виконується за незбігу жодної з умов **(**<вирази **ELSE**> **;)**.

Приклад:

```
CASE byLeft/2 OF
    0,127:
        bReset : = TRUE;
        Varl := 0;
            16..24:
                Varl := 1;
                    ELSE
                Varl := 2;
END_CASE
```

Значеннями вибору **CASE** можуть бути тільки цілі константи, змінні застосовувати не можна. Однакові значення в альтернативах вибору задавати не можна, навіть у діапазонах. Безумовно, оператор **CASE** «слабкіше» оператора **IF**, що не має подібних обмежень.

2.6 Цикл FOR

Цикл **FOR** забезпечує задану кількість повторень групи виразів. Синтаксис:

FOR <Цілий лічильник> **:=** <Початкове значення>
TO <Кінцеве значення>
[BY <Крок>**] DO**
<Вирази – тіло циклу>
END_FOR

Перед виконанням циклу лічильник одержує початкове значення. Далі тіло циклу повторюється, поки значення лічильника не перевищить кінцевого значення. Лічильник збільшується в кожному циклі. Початкове й кінцеве значення й крок можуть бути як константами, так і виразами.

Лічильник змінюється після виконання тіла циклу. Тому якщо задати кінцеве значення менше початкового, то за позитивного збільшення цикл не буде виконаний жодного разу. За однакових початкового й кінцевого значень тіло циклу буде виконано один раз.

Частина конструкції **BY** у дужках необов'язкова, вона визначає крок збільшення лічильника. За замовчуванням лічильник збільшується на одиницю в кожній ітерації. Як лічильник можна використати змінну будь-якого цілого типу. Приклад:

```
    Var1 := 1
FOR Counter: =1 TO 5 BY 1 DO
    Var1: = Var1*2;
END_FOR;
    Erg: =Var1;
```

У цьому прикладі за початкового значення **Var1,** що дорівнює **1**, після виконання циклу ця змінна буде дорівнювати **32**.

Крок зміни лічильника ітерацій може бути й негативним. Початкова умова в цьому випадку повинна бути більше кінцевої. Цикл буде закінчений, коли значення лічильника стане менше кінцевого значення.

Наприклад:

```
Varl := 0;
FOR ci := 10 TE 1 BY -1 DO
```

Varl := Varl + 1;
END_FOR

Цикл **FOR** винятково зручний для ітерацій із заздалегідь відомим числом повторів. Для побудови правильного циклу досить дотримуватися двох простих формальних вимог:

- не змінювати лічильник циклу й умови закінчення в тілі циклу. Лічильник і змінні, які утворюють кінцеву умову в циклі, можна використати тільки для читання;

- не задавати в якості кінцевої умови максимальне для визначеного типу змінної лічильника значення. Так, якщо для однобайтного цілого без знака задати константу 255, то умову закінчення не буде виконано ніколи. Цикл стане нескінченним.

2.7 Цикли WHILE й REPEAT

Цикли **WHILE** й **REPEAT** забезпечують повторення групи виразів, поки вірний умовний логічний вираз. Якщо умовний вираз завжди істина, то цикл стає нескінченним.

Синтаксис **WHILE**:

WHILE <Умовний логічний вираз**> DO**
    <Вирази — тіло циклу**>**
END_WHILE

Умова в циклі **WHILE** перевіряється до початку циклу. Якщо логічний вираз первісно має значення **FALSE (НЕПРАВДА)**, тіло циклу не буде виконано жодного разу.

Синтаксис **REPEAT**:

REPEAT
    <Вирази — тіло циклу >
        UNTIL <Умовний логічний вираз>
END_REPEAT

Умова в циклі **REPEAT** перевіряється після виконання тіла циклу. Якщо логічний вираз первісно має значення **FALSE (НЕПРАВДА)**, тіло циклу буде виконано один раз.

Приклад:

```
ci := 64;
WHILE ci > 1 DO
        Varl := Varl + 1;
                ci := ci/2;
END_WHILE
```

Правильно побудований цикл **WHILE** або **REPEAT** обов'язково повинен виконувати зміну змінних, які складають умову закінчення в тілі циклу, поступово наближаючись до умови завершення. Якщо цього не зробити, цикл не закінчиться ніколи.

Намагайтеся не використовувати точну рівність і нерівність для припинення циклу. Інакше існує ймовірність помилково проскочити граничні умови. Краще використовувати умови більше й менше.

Для реалізації мінімального часу виконання циклу необхідно уникати в тілі циклу й в умовному виразі обчислень, які можна було зробити заздалегідь. Такі обчислення повторюються в циклі, щоразу забираючи час. Наприклад:

```
WHILE ci < 5 + x DO
        Var := Varl + 2*x*x + 1;
                ci :- ci + 1;
END_WHILE
```

Даний цикл можна оптимізувати за швидкістю:

```
iMax := 5+x;
iPoly := 2*x*x + 1;
WHILE ci < iMax DO
        Var := Varl + iPoly;
                ci :- ci + 1;
END_WHILE
```

Цикл **REPEAT** відрізняється від циклу **WHILE** тим, що перша перевірка умови виходу із циклу здійснюється, коли цикл уже виконався **1** раз. Це означає, що незалежно від умови виходу цикл виконується хоча б один раз.

Зауваження. Програміст повинен бути впевнений, що цикл не стане нескінченним. Для цього в тілі циклу значення змінної, яка визначає вхідну умову, обов'язково повинне змінюватися. Наприклад, шляхом інкремента або декремента лічильника.

Приклад:

```
WHILE counter<>0 DO
    Var1: = Var1*2;
        counter: = counter-1;
END_WHILE
```

2.8 Переривання ітерацій операторами EXIT й RETURN

Оператор **EXIT**, який розташований у тілі циклів **WHILE**, **REPEAT** й **FOR**, приводить до негайного закінчення циклу. Розглянемо, наприклад, пошук елемента масиву з певним значенням (**x)**. Найпростіше виконати лінійний перебір за допомогою циклу **FOR**:

```
bObtained:= FALSE;
FOR c := 1 TO Maxlndex DO
    IF x - a[c] THEN
        Index := c;
        bObtained := TRUE;
        EXIT;
    END_IF
END_FOR
IF bObtained THEN (*елемент знайдено, його індекс – Index*)
```

Для вкладеного циклу оператор **EXIT** завершує тільки «свій» цикл, зовнішній цикл буде продовжувати роботу. Наприклад:

```
FOR в := ОБ ТЕ 9 DO
    FOR x := ОБ ТЕ 99 DO (* обробляємо рядок масиву Arr[в][x]*)
    ;
        IF … THEN EXIT; (*'хвіст' рядка обробляти не треба, переходимо
    до наступного*)
    END_FOR
END_FOR
```

За необхідності завершення зовнішнього циклу за умови, яка виникла у вкладеному циклі, можна використати пару синхронізованих операторів **EXIT**:

```
bBreak := FALSE;
FOR в := ОБ ТЕ 9 DO
    FOR x := 0 TO 99 DO
        IF _ THEN bBreak :** TRUE; EXIT; (*перервати обробку*)
    END_FOR
```

```
        IF bBreak THEN EXIT;
END__FOR
```

Оператор **RETURN** здійснює негайне повернення з **POU**. Це єдиний спосіб перервати вкладені ітерації без введення додаткових перевірок умов.

2.9 Ітерації на базі робочого циклу ПЛК

Використовувати для умови виходу із циклів **WHILE** і **REPEAT** входи, виходи або інші апаратно-залежні змінні ПЛК не можна. Дані змінні не змінюють своїх значень у межах одного робочого циклу користувальницької програми, тому цикл завжди буде нескінченним. Якщо подібна необхідність все-таки є, використайте для ітерацій робочий цикл ПЛК. Виконання ітерації задається простою умовою **IF**. Аналогічно можна поступати за необхідності побудови тривалих циклів. Наприклад, ініціалізація або копіювання більших масивів даних. «Розмазування» об'ємних операцій на безліч робочих циклів є стандартним прийомом, що дозволяє уникнути небажаного вповільнення інших, паралельно виконуваних завдань.

Так, ініціалізація даних функціонального блока з виставлянням сигналу готовності може виглядати так:

```
ENO:          BOOL: = FALSE;
niCounter: INT: = 1000;
aiVar: ARRAY [0.999] OF INT;
IF niCounter = 0 THEN
        ENO: = TRUE;
        …      (*основна робота блока*)
        ELSE
                niCounter: = niCounter – 1;
                aiVar [niCounter] := GetlnitVal(niCounter);
                                        (*складна ініціалізація*)
END_IF
```

2.10 Оформлення тексту

Оформлення текстів **ST**-програм може бути зовсім довільним. Розташування операторів і виразів у рядку не впливає на правильність програм. Але дуже важливо виробити свій власний стиль і строго дотримуватися його. Найважливішу роль в оформленні відіграють відступи на початку рядків. Відступи зорово поєднують рядки, що містять вираз одного рівня вкладення. Текст, вирівняний у вигляді драбинки, кожна сходинка якої ставиться до одного циклу або умови, читати легко. Незважаючи на можливість горизонтального прокручування в редакторі, бажано, щоб по ширині текст містився на одній сторінці. Не варто розташовувати кілька виразів в один рядок. Нічого поганого

немає в тім, що текст виявиться розтягнутим по вертикалі: лаконічні вирази й навіть порожні рядки тільки допомагають зоровому аналізу.

Для оформлення **ST** текстів цілком застосовні рекомендації, які можна зустріти в літературі за програмування на **Паскалі** й **C**. Зверніть увагу, що в **ST** відсутні горезвісні програмні дужки (у **Паскалі: begin, end; у C: {}**). Замість них кожний вираз мови має власну кінцівку (**WHILE .. END_WHILE, IF .. END_IF**). Тобто закриваюча програмна дужка є інформативною. На вигляд такий текст сприймається явно краще. При створенні складних вкладень у мові **C** дужки, які є закриваючими, часто розташовані суцільною драбинкою. У таких випадках досвідчені програмісти застосовують короткі коментарі після кожної дужки, яка є закриваючою. Коментарі підказують, із чого початий даний рівень відступу. Наприклад: (***FOR x***). Це хороший прийом, але при грамотному застосуванні відступів у рядках **ST** така необхідність виникає значно рідше, ніж у **C** й **Паскалі**.

3 Контрольні питання

1 Поясніть різницю між циклом **WHILE** і **REPEAT**.
2 Як працює інструкція **IF** і її призначення?
3 Який формат даних має лічильник циклів?
4 Для чого призначений цикл **FOR**?

4 Домашнє завдання

4.1 На підставі розробленого алгоритму керування ділянки виробничого процесу й обраної викладачем конкретної частини розробити його програмну реалізацію мовою ST.

4.2 Підготувати протокол згідно з розд. 6.

5 Лабораторне завдання

5.1 Запустити середовище розробки **CoDeSys**. Якщо в організаторі об’єктів автоматично відкривається проект, потрібно закрити його за допомогою меню **Файл → Закрити (File → Close)** [2, 3].

5.2 Відкрити новий проект за допомогою кнопки **New (Створити)** на панелі інструментів або меню **Файл → Відкрити (File → Open)**. У вікні, що відкрилося, **Настроювання цільової платформи (Target Setting)** виберіть **None.** У новому вікні, що відкрилося, **Новий програмний компонент (New POU)** виберіть тип **POU «Програма»** і мову програмування **ST**. Натисніть кнопку **ОК**.

5.3 В робочій області, що з'явилася, за допомогою клавіатури ввести програму, яка виконує алгоритм логічного керування ділянки технологічного процесу згідно із домашнім завданням. Додати коментарі для окремих частин програми. При написанні програми слід визначати змінні, які характеризують змістове значення параметра і правильно вибрати та тип змінних. Зберегти

проект під унікальним ім'ям. Занести до протоколу перелік програм **(PRG)**, які входять у даний проект. Цей перелік відображається в організаторі об'єктів.

5.4 Ввімкнути режим симуляції за допомогою меню **Онлайн → Режим сімуляції (Online → Simulation Mode)**.

5.5 Під'єднати середовище до симуляції ПЛК за допомогою меню **Онлайн → Підключення (Online → Login)**. Зверніть увагу на те, що під час під'єднання виконується компіляція проекту.

5.6 Запустити проект на виконання за допомогою меню **Онлайн → Старт (Online → Run)** та виконати його тестування на правильність виконання.

5.7 Після тестування записати його в зошиті для лабораторних робіт.

6 Зміст протоколу

Протокол лабораторної роботи "Програмування ПЛК. Вивчення мови структурований текст (ST)" оформлюється в окремому зошиті для лабораторних робіт. Протокол має містити назву лабораторної роботи та її мету, відповіді на контрольні питання, хід виконання домашнього та лабораторного завдання, коментарі до програм розроблених та налагоджених під час виконання лабораторної роботи, блок-схеми програм, висновки.

7 Перелік рекомендованої літератури

1. Петров И.В Программируемые контроллеры. Стандартные языки и приемы прикладного программирования / Под ред. проф. В.П. Дьяконова. – М: ООО «СОЛОН-Пресс», 2004. – 256 с.
2. Методичні посібники для лабораторних робіт даного курсу.
3. Конспект лекцій з курсу МП і ПЗА.
4. Веб-сторінка фірми Smart Software Solutions Gmb виробника середовища CoDeSys http://www.3s-software.com/
5. Веб-сторінка ПК "Пролог", підтримка середовища CoDeSys російською мовою http://www.codesys.ru/
6. Христенсен, Дж. Х. Знакомство со стандартом на языки программирования PLC: IEC 1131-3 (МЭК 1131-3) [Электронный ресурс]/ Дж. Х. Христенсен. – [2004]. – Режим доступа: http://www.asutp.ru
7. Руководство пользователя по программированию ПЛК в CoDeSys V2.3. – Смоленск: ПК "Пролог", 2004. – 423 с.
8. Деменков Н.П. Языки программирования промышленных контроллеров: Учебное пособие / Под ред. К.А. Пупкова.– М.: Изд-во МГТУ им. Н.Э. Баумана. – 172 с.

Навчально-методичне видання

Тігарєв А.М.

Методичні вказівки
до виконання лабораторної роботи
“Програмування ПЛК.
Вивчення мови структурований текст (ST)”

Редактор *Кодрул Л.А.*
Комп’ютерне верстання *Гардиман Ж.А.*

Підписано до друку 26.06.2017.
Формат 60/88/16 Обсяг 0,9 друк. арк.
Тираж 25 прим. Зам. № 07/17.
Віддруковано в редакційно-видавничому центрі ОНАЗ ім. О.С. Попова
м. Одеса, вул. Ковалевського, 5
Тел. 7050 494